Advanced Network Technologies

Multimedia 1/2

Dr. Wei Bao| Lecturer
School of Computer Science

THE UNIVERSITY OF SYDNEY

Drag picture to placeholder or clic

Multimedia

- Multimedia
- Streaming stored video
- Voice-over-IP
- RTP/SIP

Multimedia

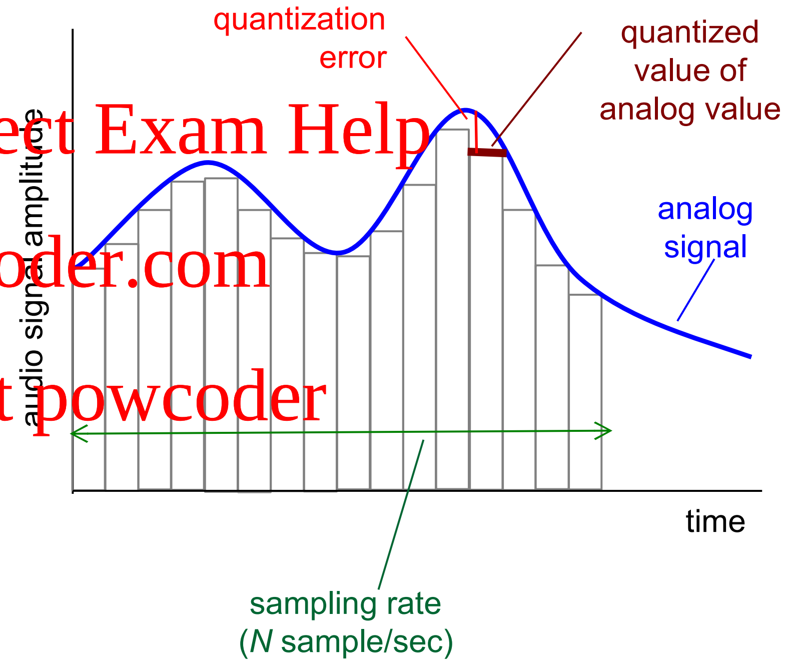
THE UNIVERSITY OF SYDNEY

Multimedia networking: 3 application types

- *streaming, stored* audio, video
  - *streaming:* can begin playout before downloading entire file
  - *stored (at server):* can transmit faster than audio/video will be rendered (implies storing/buffering at client)
  - e.g., YouTube, Netflix, Hulu
- *conversational* voice/video over IP
  - interactive nature of human-to-human conversation limits delay tolerance
  - e.g., Skype
- *streaming live* audio, video
  - e.g., live sporting event

Multimedia audio

- analog audio signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
  - e.g., 2^8=256 possible quantized values
  - each quantized value represented by bits, e.g., 8 bits for 256 values



Rate=44100 samples/sec * 8bit/sample = 352800 bps

Video

- Video: sequence of images displayed at constant rate
  - e.g. 24 images/sec
- Each image: array of pixels: Resolution: e.g. 480*640
  - each pixel: 3 colors
  - Red, Green, Blue (RGB)
  - Each color has 2^8=256 possible quantized values (8 bit)
  - Data rate: 8*3*480*640*24 = 177 Mbps. Too large!

- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)

frame *i*

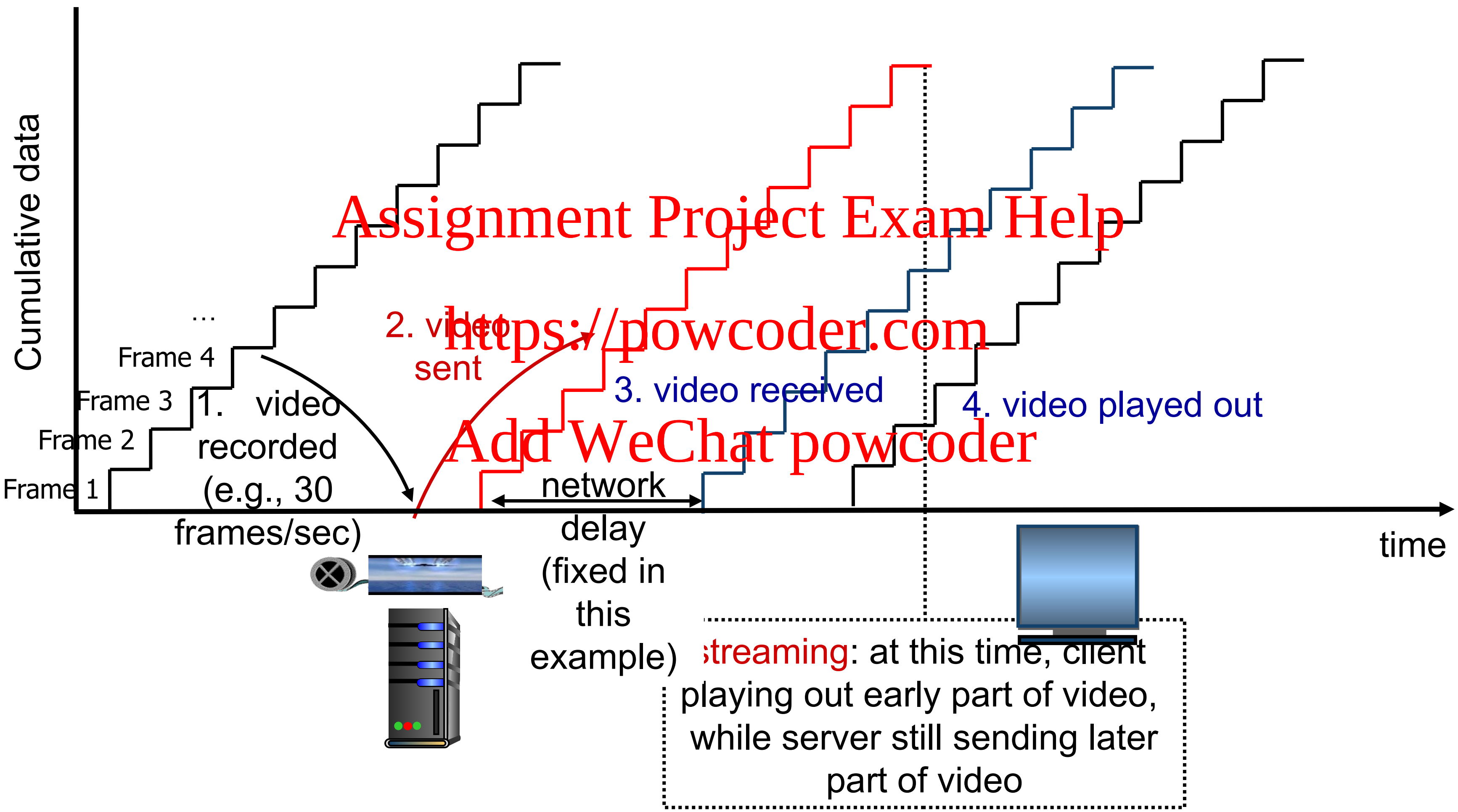
- examples:
- MPEG 1 (CD-ROM) 1.5 Mbps
- MPEG2 (DVD) 3-6 Mbps
- MPEG4 (often used in Internet, < 1 Mbps)
  - MPEG: Moving Picture Experts Group

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i+1*

Streaming Stored Video

Streaming stored video

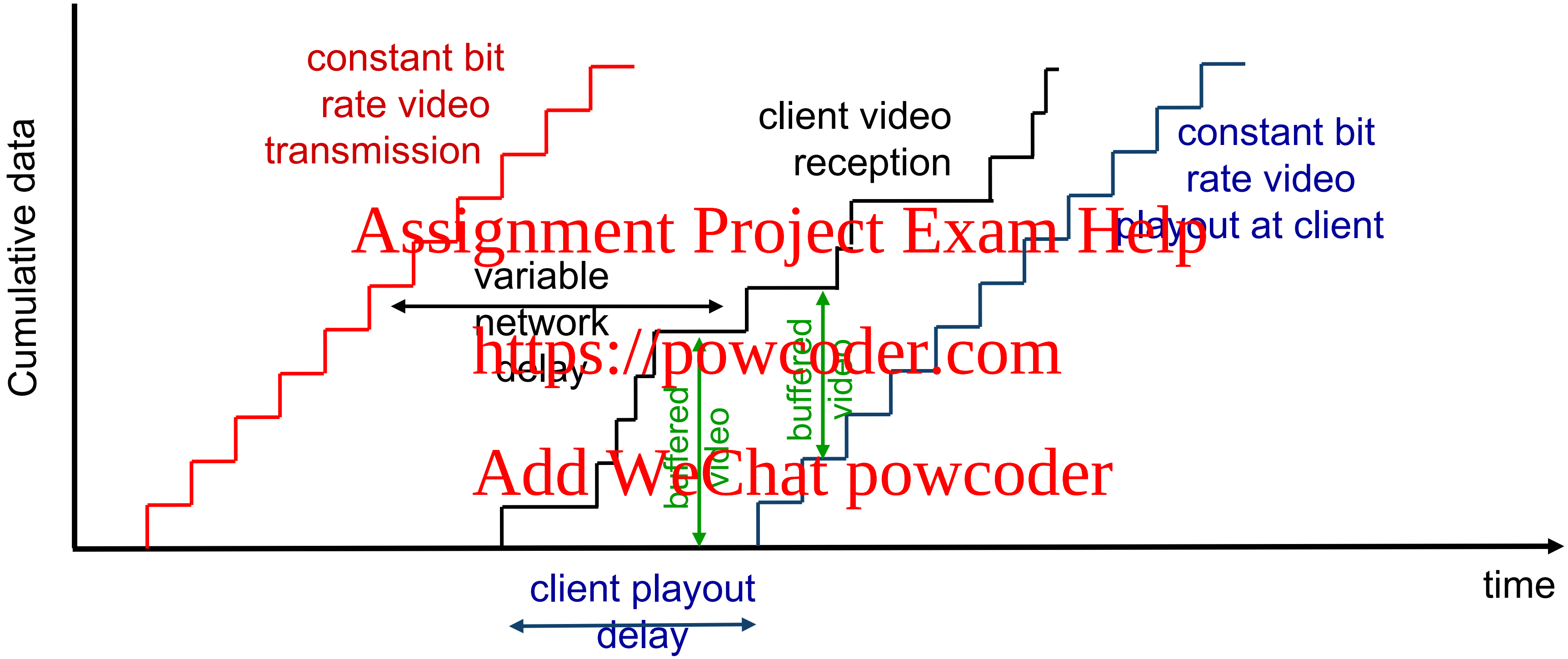


THE UNIVERSITY OF SYDNEY

Streaming stored video: challenges

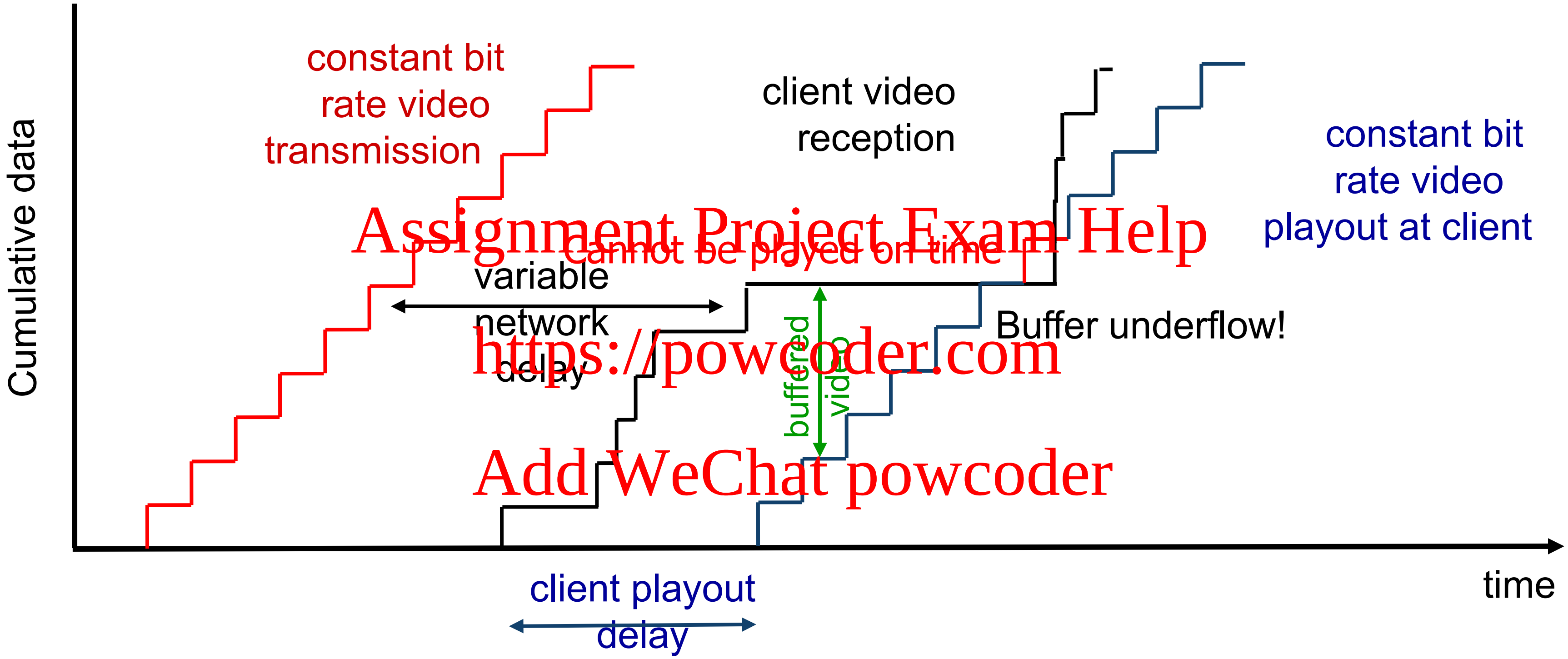
- continuous playout constraint: once client playout begins, playback must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match playout requirements
- other challenges:
  - client interactivity: pause, fast-forward, jump through video
  - video packets may be lost, retransmitted

Streaming stored video: revisited

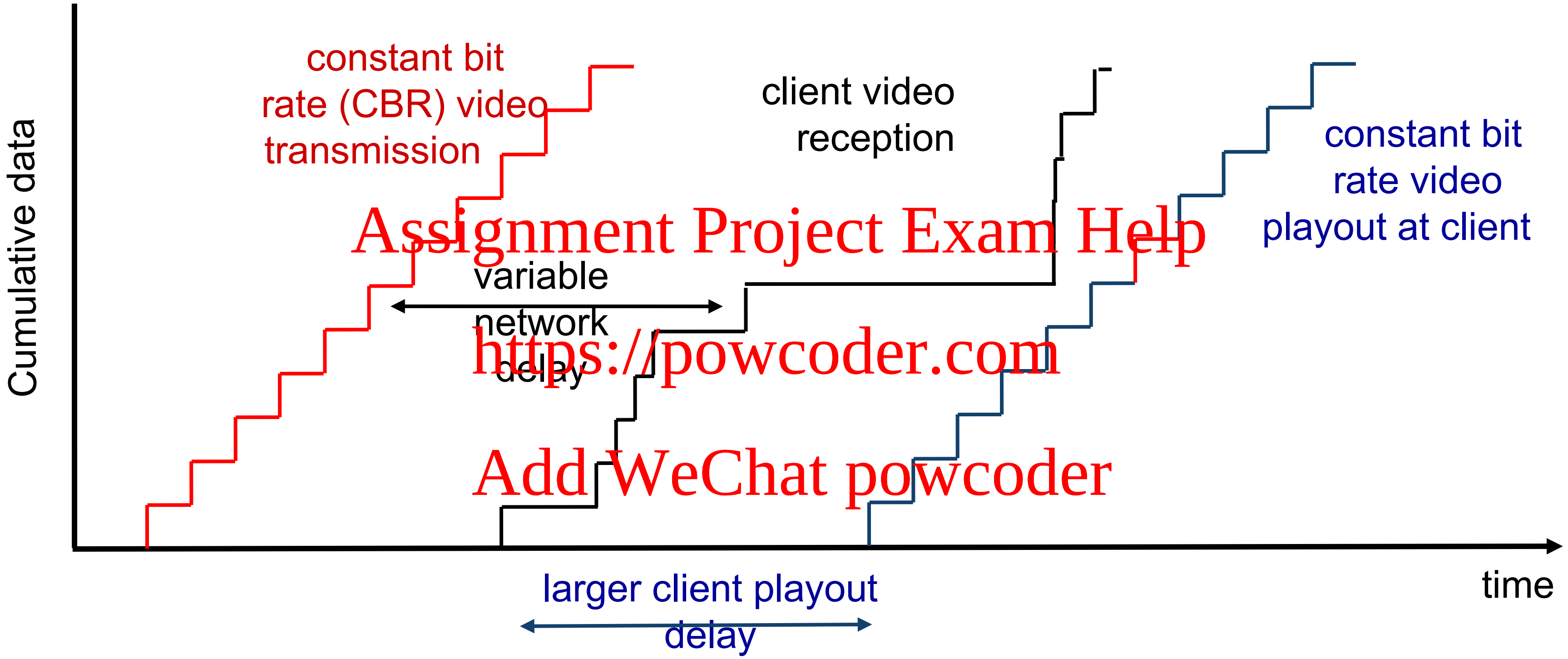


› *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Streaming stored video: revisited

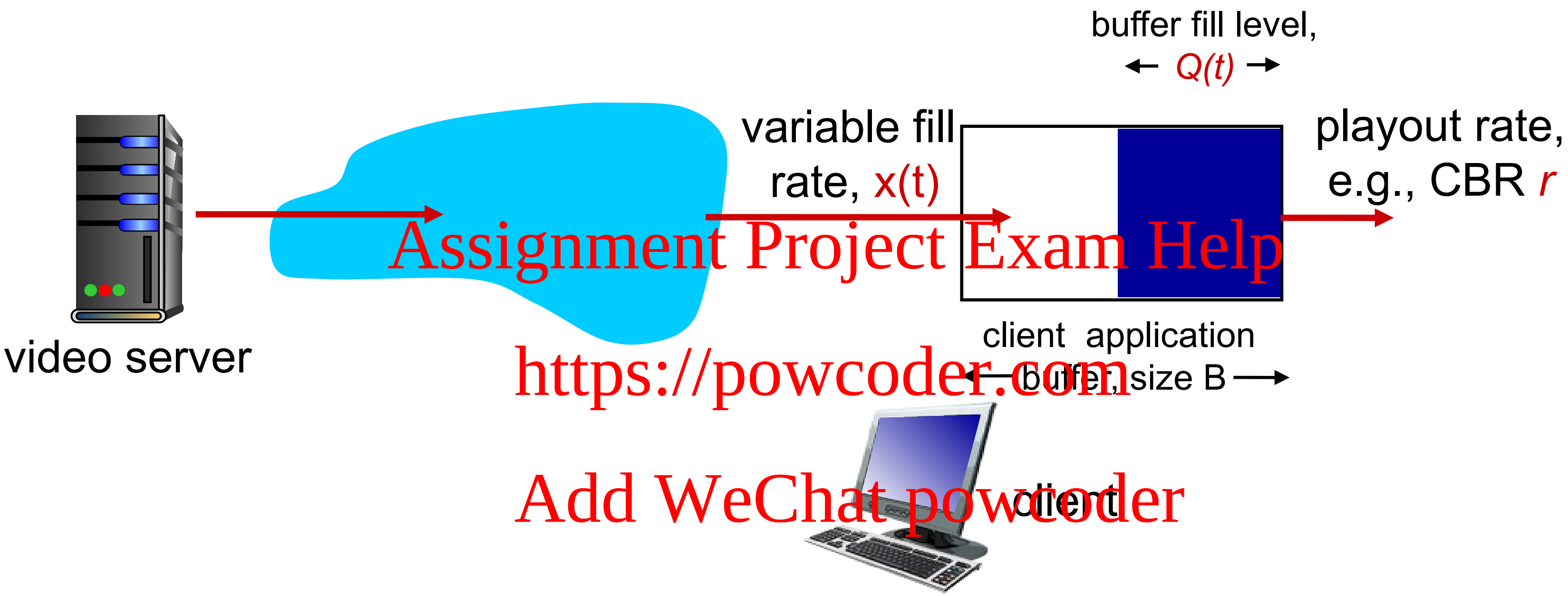


Streaming stored video: revisited



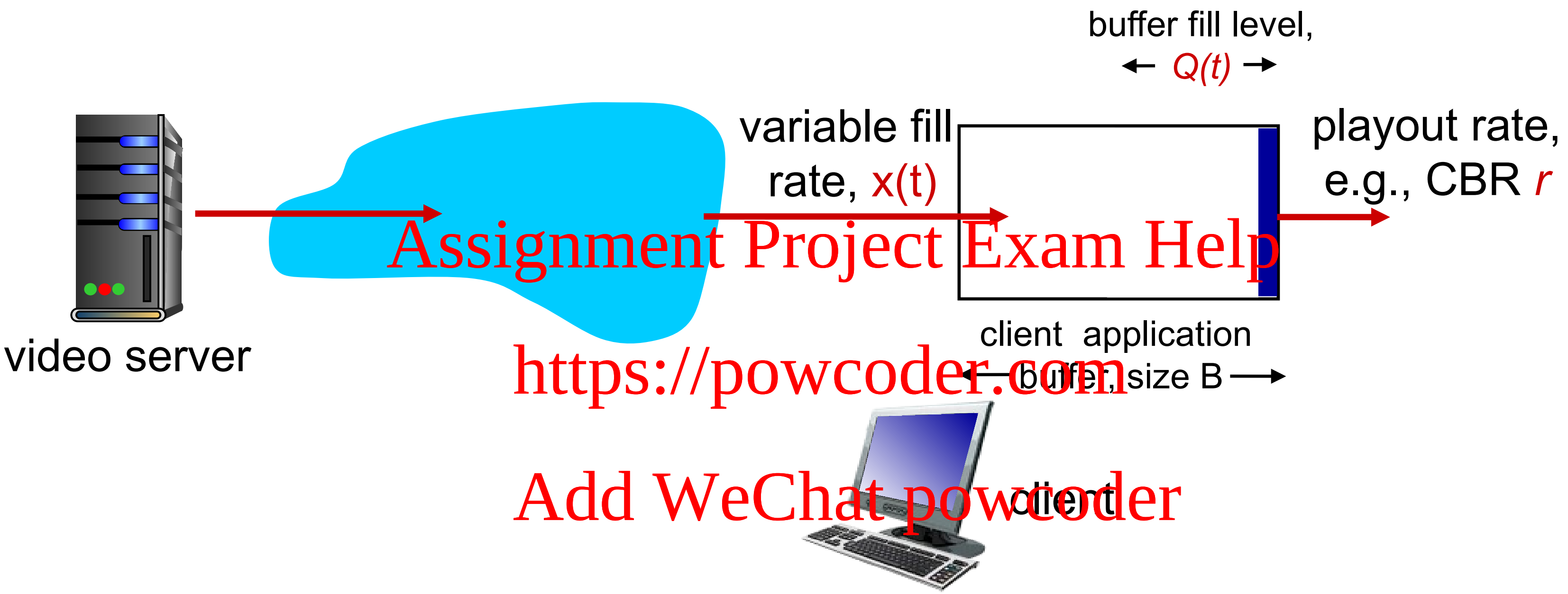
- *Increase playout delay: fewer buffer underflows*
- *initial playout delay tradeoff*

Client-side buffering, playout



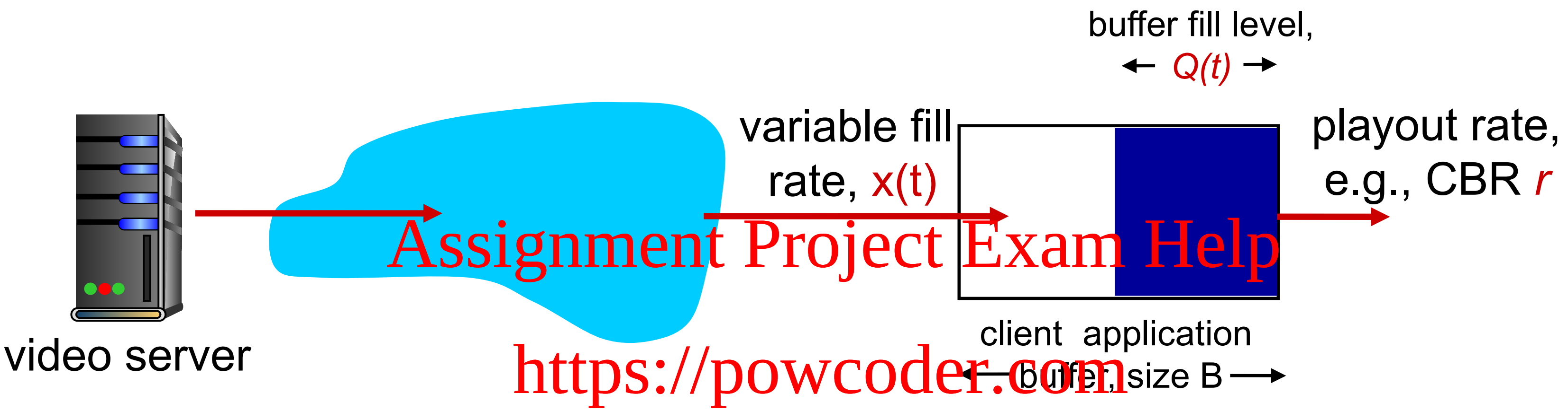
THE UNIVERSITY OF SYDNEY

Client-side buffering, playout



1. Initial fill of buffer until playout begins at t_p
2. playout begins at $t_{p,}$
3. buffer fill level Q(t) varies over time as fill rate x(t) varies and playout rate r is constant
4. $Q(t+1)=Q(t)+x(t)$, $t \leq t_{p;}$ $Q(t+1)=\max[Q(t)+x(t)-r, 0]$, $t> t_p$
5. $Q(t)+x(t)-r<0$: buffer underflow

Client-side buffering, playout



playout buffering: average fill rate E(x), playout rate r

› E(x) < r: buffer eventually empties (causing freezing of video playout until buffer fills again)

› E(x) ≥ r: buffer will not empty, provided initial playout delay is large enough to absorb variability in x(t)

- *initial playout delay tradeoff:* buffer starvation less likely with larger delay, but larger delay until user begins watching

Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate
  - transmission rate can be oblivious to congestion levels
- short playout delay (2-5 seconds) to remove network jitter
- error recovery: application-level, time-permitting
- RTP [RFC 2326]: multimedia payload types
- UDP may *not* go through firewalls

Streaming multimedia: HTTP

› multimedia file retrieved via HTTP GET

› send at maximum possible rate under TCP

› fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery)

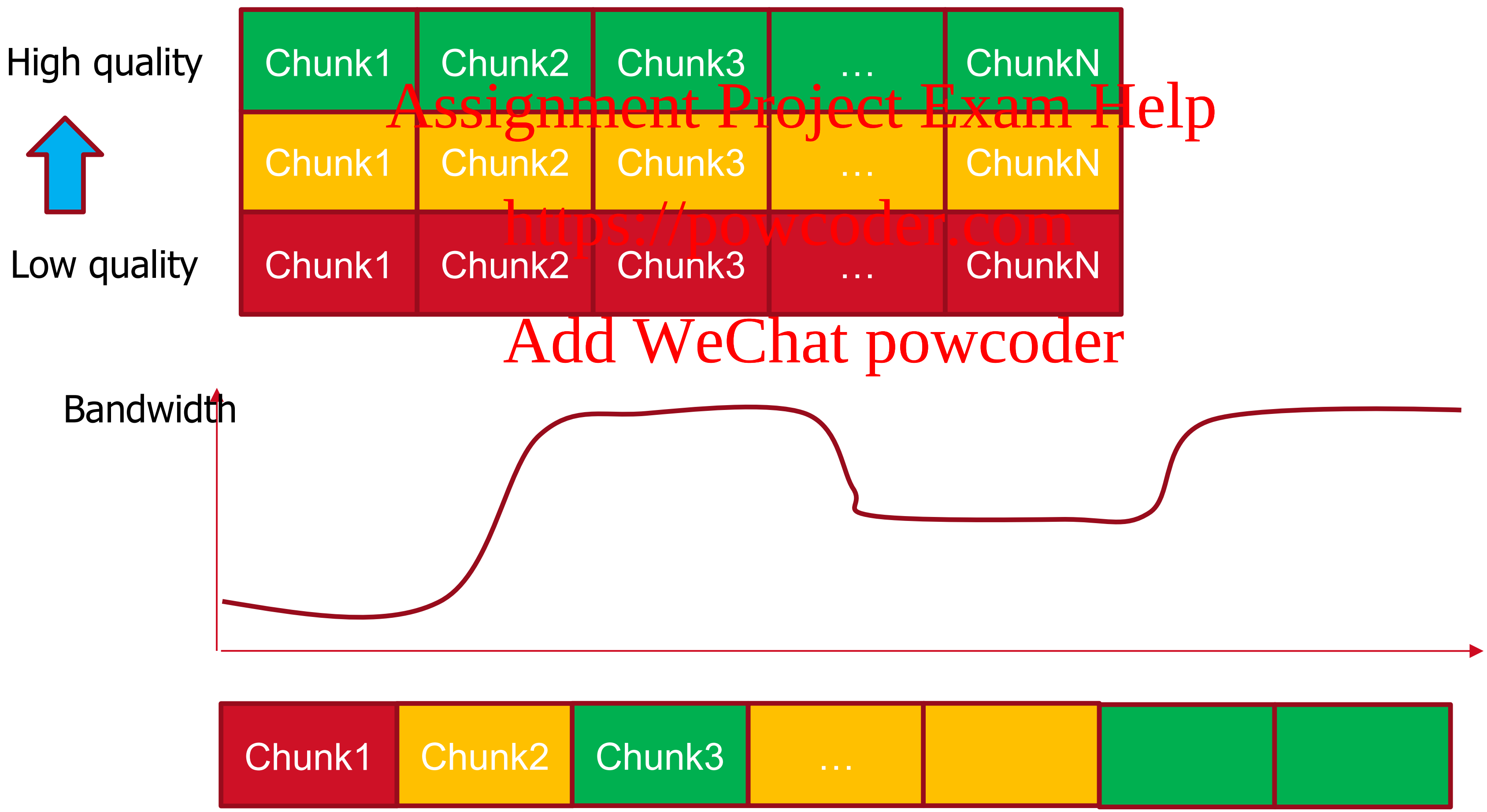
› larger playout delay: smooth TCP delivery rate

› HTTP/TCP passes more easily through firewalls

Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on current available bandwidth)

Streaming multimedia: DASH



Streaming multimedia: DASH

› DASH: Dynamic, Adaptive Streaming over HTTP

› *“intelligence”* at client: client determines

- *when* to request chunk (so that buffer starvation does not occur)
- *what encoding rate* to request (higher quality when more bandwidth available)
- *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)

Content distribution network

› *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

› *option 1:* single, large "mega-server"

- single point of failure
- point of network congestion
- long path to distant clients
- multiple copies of video sent over outgoing link

….quite simply: this solution *doesn't scale*

Content distribution network

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*

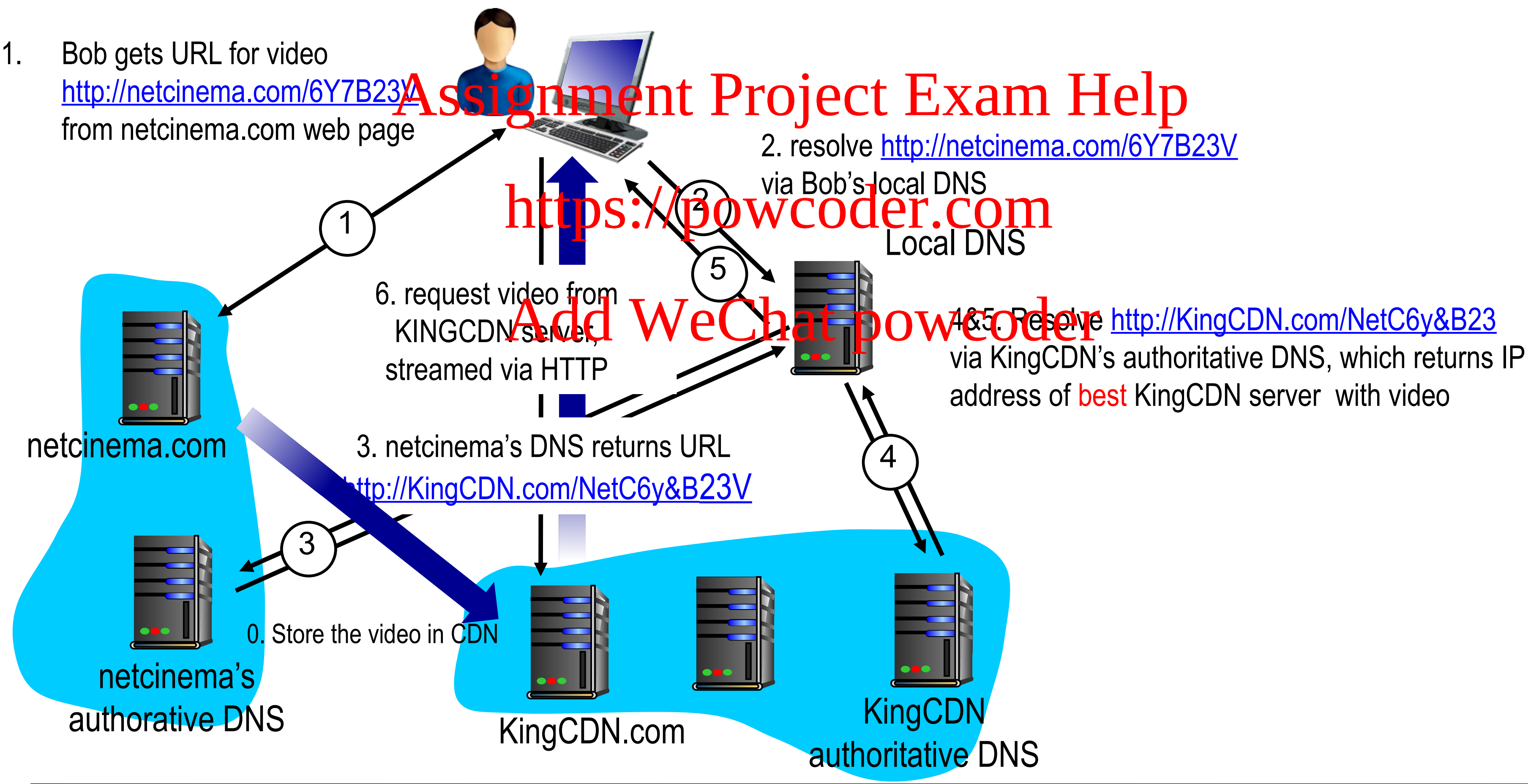
Akamai

Limelight NETWORKS

CDN: "simple" content access scenario

Bob (client) requests video http://netcinema.com/6Y7B23V

- video stored in CDN at http://KingCDN.com/NetC6y&B23V



CDN cluster selection strategy

- *challenge:* how does CDN DNS select “good” CDN node to stream to client
  - pick CDN node geographically closest to client
  - pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)

- *alternative:* let *client* decide - give client a list of several CDN servers
  - client pings servers, picks “best”
  - Netflix approach

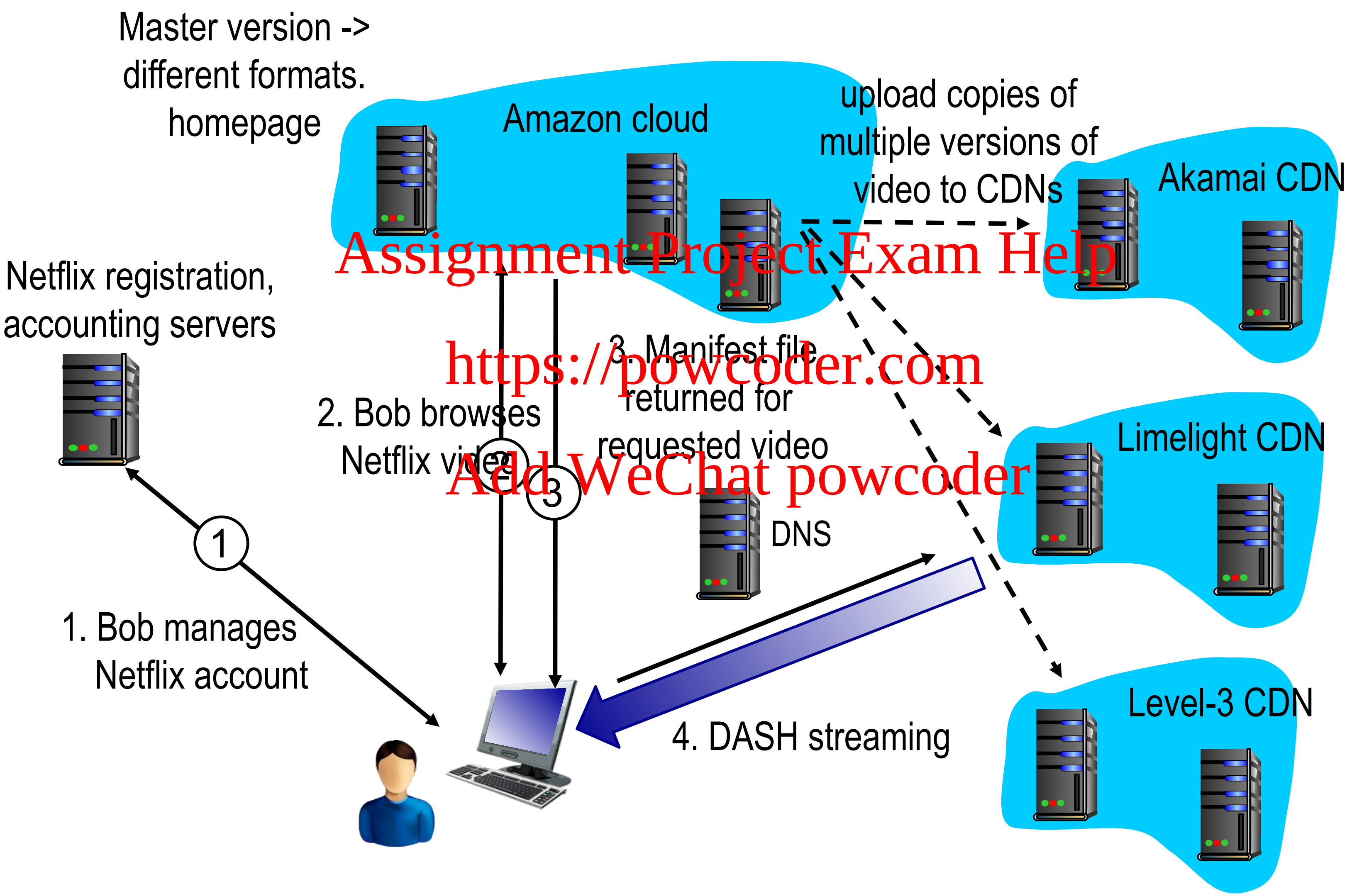
NETFLIX

THE UNIVERSITY OF SYDNEY

Case study: Netflix

- 30% downstream US traffic in 2011
- Owns very little infrastructure, uses 3rd party services:
  - own registration, payment servers
  - Amazon (3rd party) cloud services:
    - Create multiple versions of movie (different encodings) in Amazon cloud
    - Upload versions from cloud to CDNs
    - Cloud hosts Netflix web pages for user browsing
  - *three* 3rd party CDNs host/stream Netflix content: Akamai, Limelight, Level-3

Case study: Netflix



Voice over IP

THE UNIVERSITY OF SYDNEY

Voice-over-IP (VoIP)

- *VoIP end-end-delay requirement*: needed to maintain "conversational" aspect
  - higher delays noticeable, impair interactivity
  - < 150 msec: good
  - > 400 msec: bad
  - includes application-level (playout), network delays
- *session initialization:* how does callee advertise IP address, port number, encoding algorithms?
- *value-added services:* call forwarding, screening, recording
- *emergency services:* 911/000

VoIP characteristics

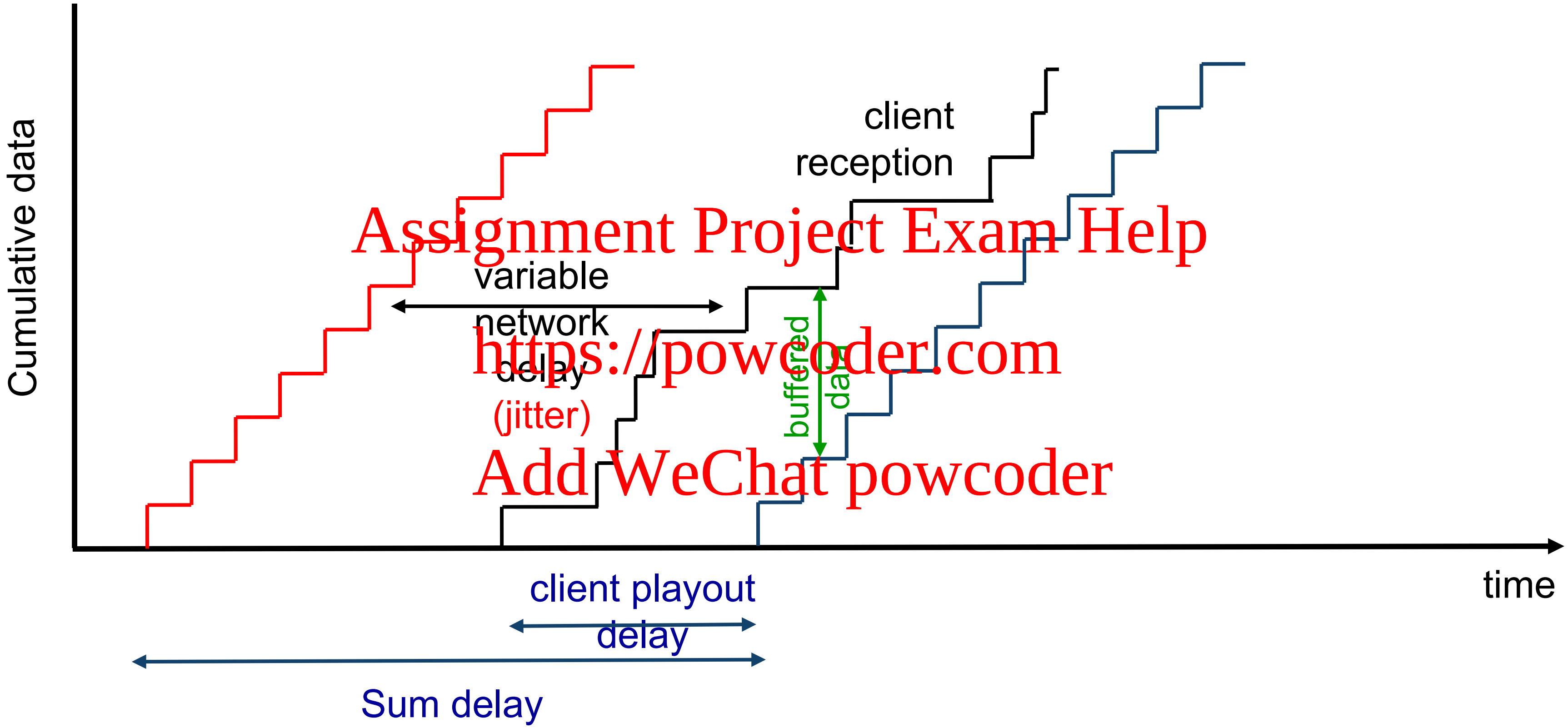
- speaker's audio: alternating talk spurts, silent periods.
  - 64 kbps during talk spurt
  - chucks generated only during talk spurts
  - 20 msec: chucks at 8 Kbytes/sec: 160 bytes of data
- application-layer header added to each chunk
- chunk+header encapsulated into UDP or TCP segment
- application sends segment into socket every 20 msec during talkspurt

THE UNIVERSITY OF SYDNEY

VoIP: packet loss, delay

- *network loss:* IP datagram lost due to network congestion (router buffer overflow)
- *delay loss:* IP datagram arrives too late for playout at receiver
  - delays: processing, queueing in network, transmission, proporgation.
  - typical maximum tolerable delay: 400 ms
- *loss tolerance:* depending on voice encoding, loss concealment, packet loss rates between 1% and 10% can be tolerated

Delay jitter



VoIP: fixed playout delay

› receiver attempts to playout each chunk exactly *q* msecs after chunk was generated.

- chunk has time stamp *t:* play out chunk at *t+q*
- chunk arrives after *t+q*: data arrives too late for playout: data "lost"

› tradeoff in choosing *q*:

- *large q:* less packet loss
- *small q:* better interactive experience

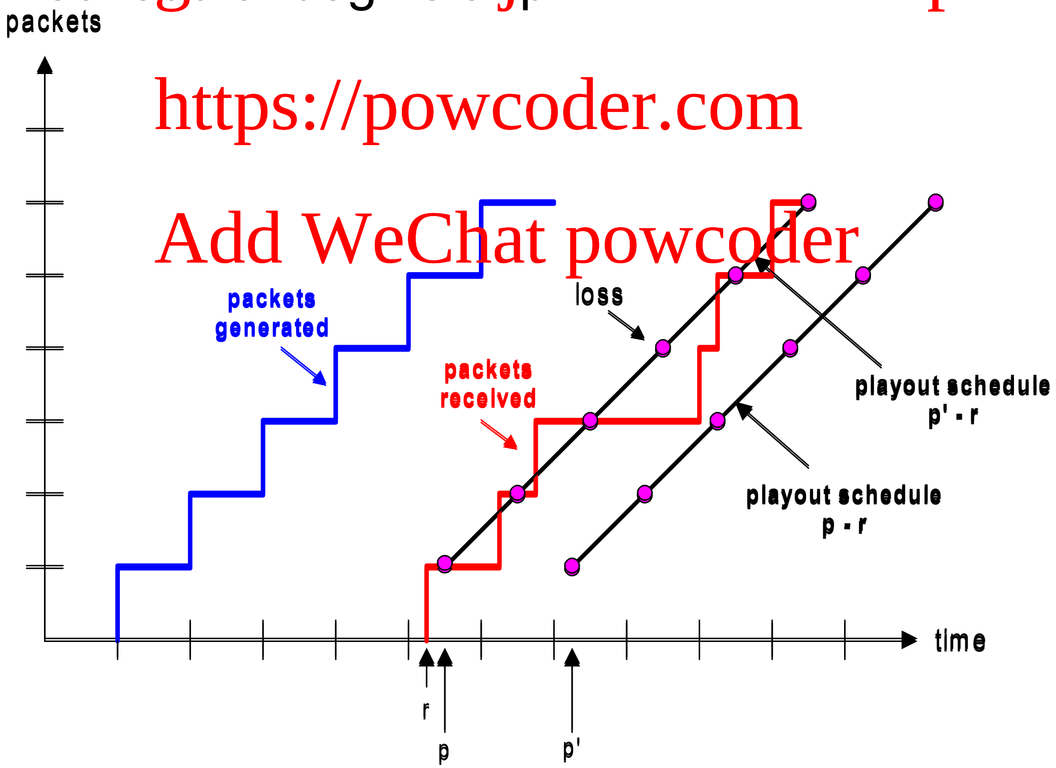
VoIP: fixed playout delay

› sender generates packets every 20 msec during talk spurt.

› first packet received at time r

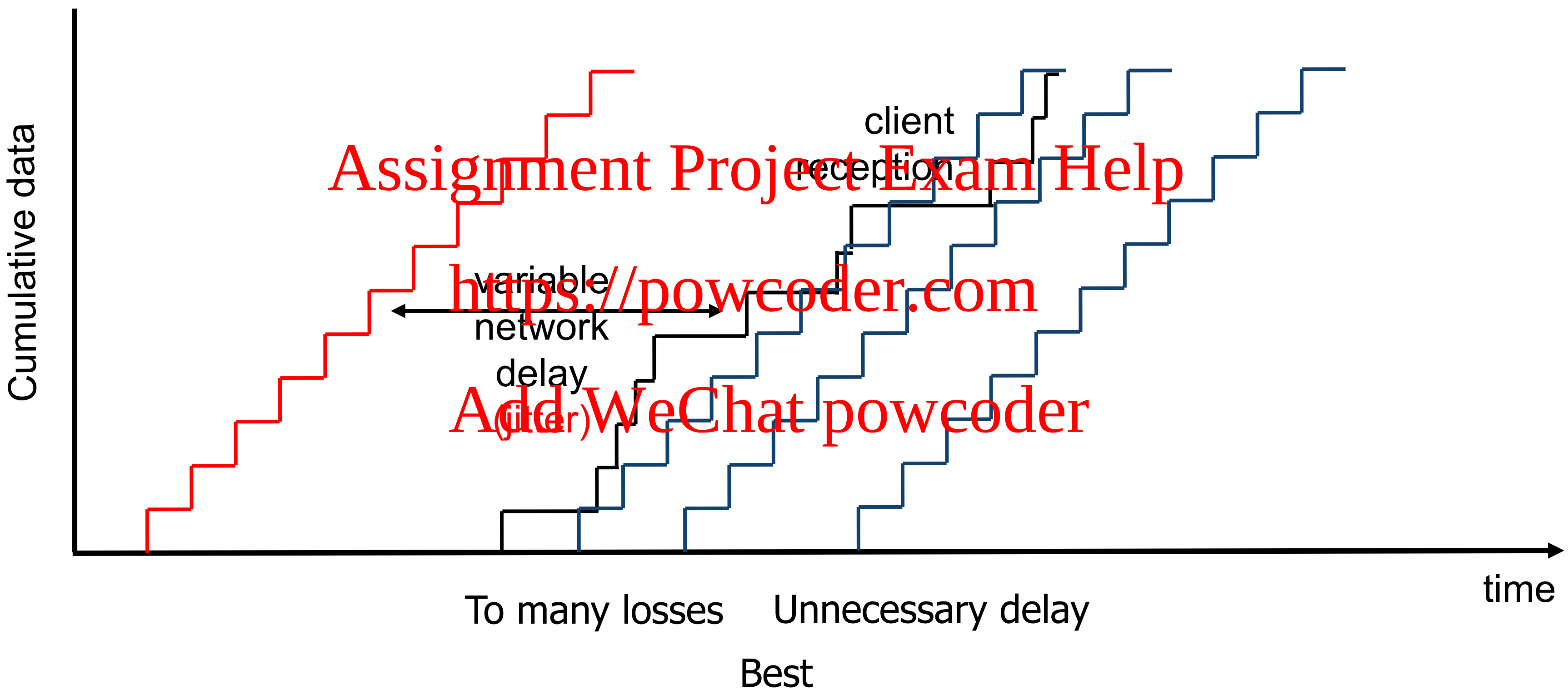
› first playout schedule: begins at p

› second playout schedule: begins at p'



Adaptive playout delay

- *goal:* low playout delay, low late loss rate



Adaptive playout delay

- *goal:* low playout delay, low late loss rate
- *approach:* adaptive playout delay adjustment:
  - estimate network delay, adjust playout delay at beginning of each talk spurt
  - silent periods compressed and elongated
- adaptively estimate packet delay: (EWMA - exponentially weighted moving average, recall TCP RTT estimate):

$$d_i = (1-\alpha)d_{i-1} + \alpha\,\underbrace{(\boldsymbol{r_i} - \boldsymbol{t_i})}_{\text{measured delay of ith packet}}$$

delay estimate after ith packet

small constant, e.g. 0.01

time received - time sent (timestamp)

measured delay of ith packet

Adaptive playout delay (cont'd)

- also useful to estimate average deviation of delay, v_i :

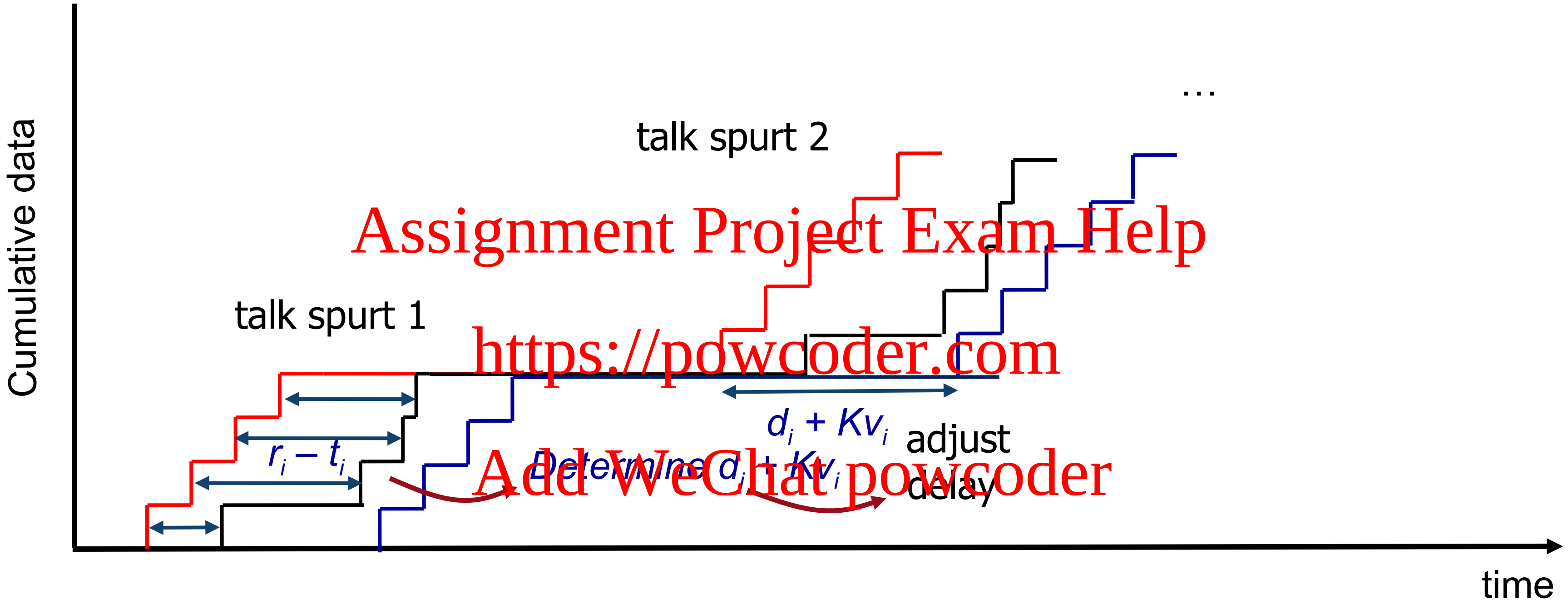
$$v_i = (1-\beta)v_{i-1} + \beta\,|r_i - t_i - d_i|$$

- estimates d_i, v_i calculated for every received packet, but used only at start of talk spurt

- for first packet in talk spurt, playout time is:

$$playout\text{-}time_i = t_i + d_i + Kv_i$$

Delay jitter



THE UNIVERSITY OF SYDNEY

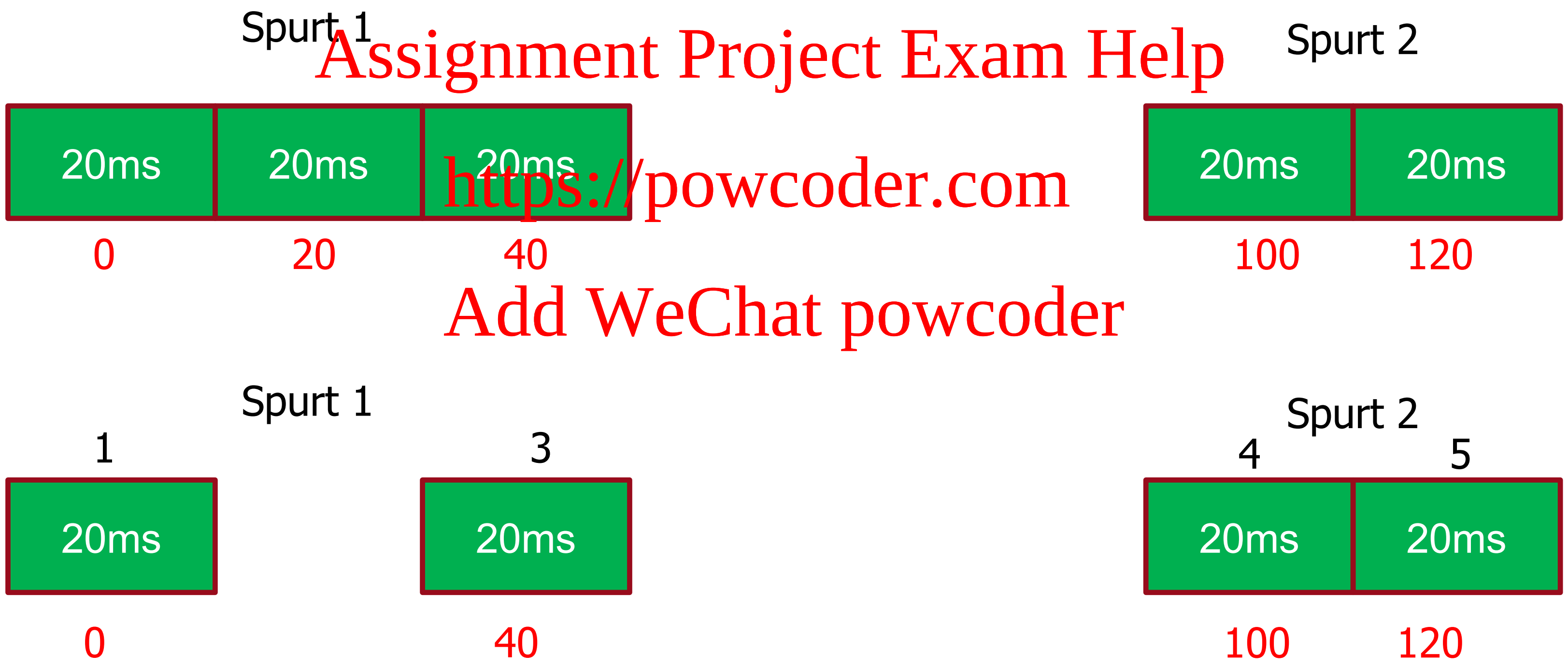
Adaptive playout delay (cont'd)

Q: How does receiver determine whether packet is first in a talkspurt?

- if no loss, receiver looks at successive timestamps
  - difference of successive stamps > 20 msec $\Rightarrow$ talk spurt begins.

- with loss possible, receiver must look at both time stamps and sequence numbers
  - difference of successive stamps > 20 msec *and* sequence numbers without gaps $\Rightarrow$ talk spurt begins.

Adaptive playout delay (cont'd)



VoIP: recovery from packet loss

Challenge: recover from packet loss given small tolerable delay between original transmission and playout

- each ACK/NAK takes ~ one RTT
- alternative: *Forward Error Correction (FEC)*
  - send enough bits to allow recovery without retransmission

simple FEC

- for every group of *n* chunks, create redundant chunk by exclusive OR-ing *n* original chunks
- send *n+1* chunks, increasing bandwidth by factor *1/n*
- can reconstruct original *n* chunks if at most one lost chunk from *n+1* chunks
- Send x_1, x_2, x_3, … x_n, and $y=x_1$ xor x_2 xor x_3,…, xor x_n,   1 0 1 0
- If x_3 is lost, can re-compute x_3 from x_1, x_2, x_4, … x_n, and y

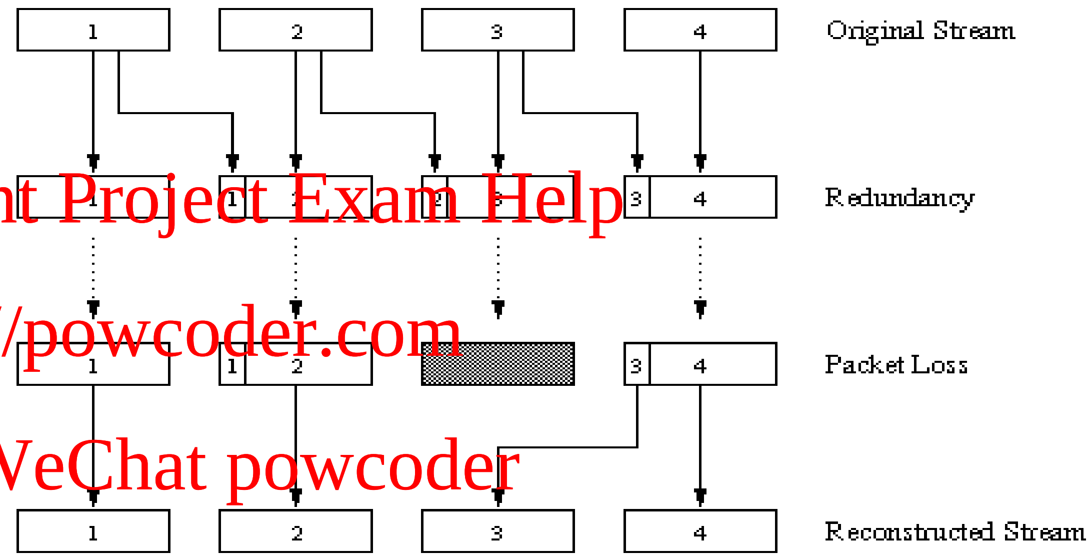
1 0 ? 0   1 XOR 0 XOR x_3 =0 x_3 =1

THE UNIVERSITY OF SYDNEY

VoIP: recovery from packet loss (cont'd)

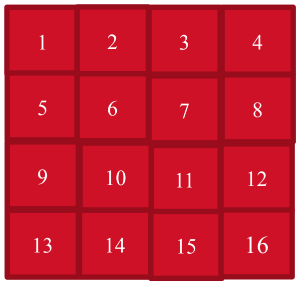
another FEC scheme:

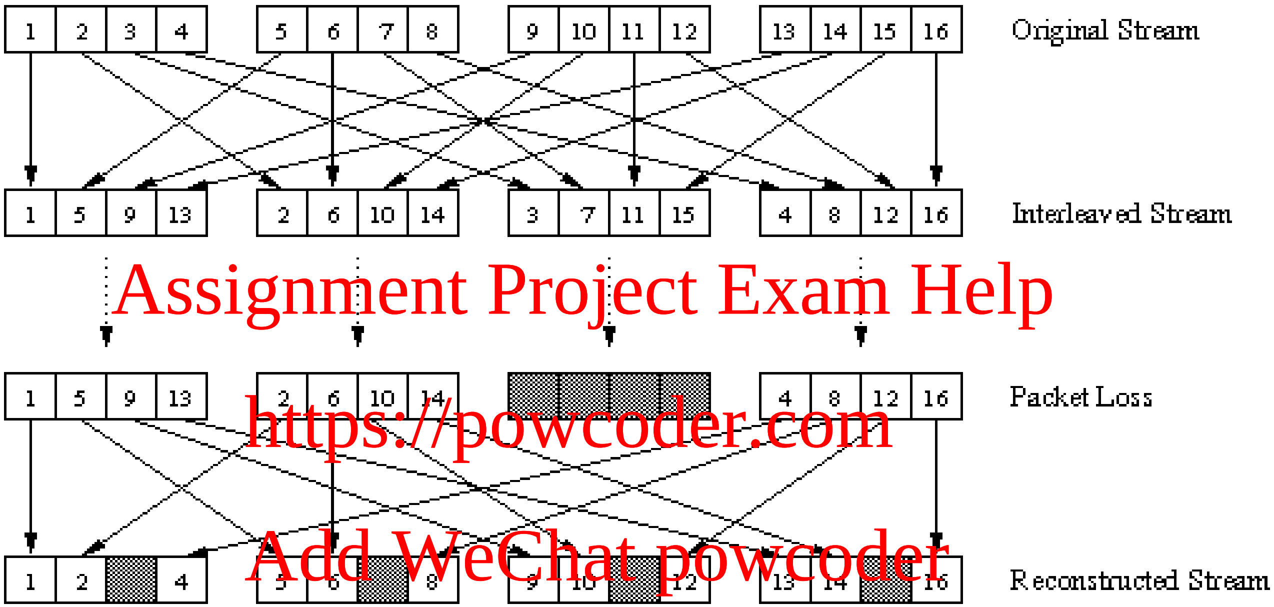
- "piggyback lower quality stream"
- send lower resolution audio stream as redundant information
- e.g., nominal stream at 64 kbps and redundant stream at 13 kbps



THE UNIVERSITY OF SYDNEY

VoiP: recovery from packet loss (cont'd)





interleaving to conceal loss:

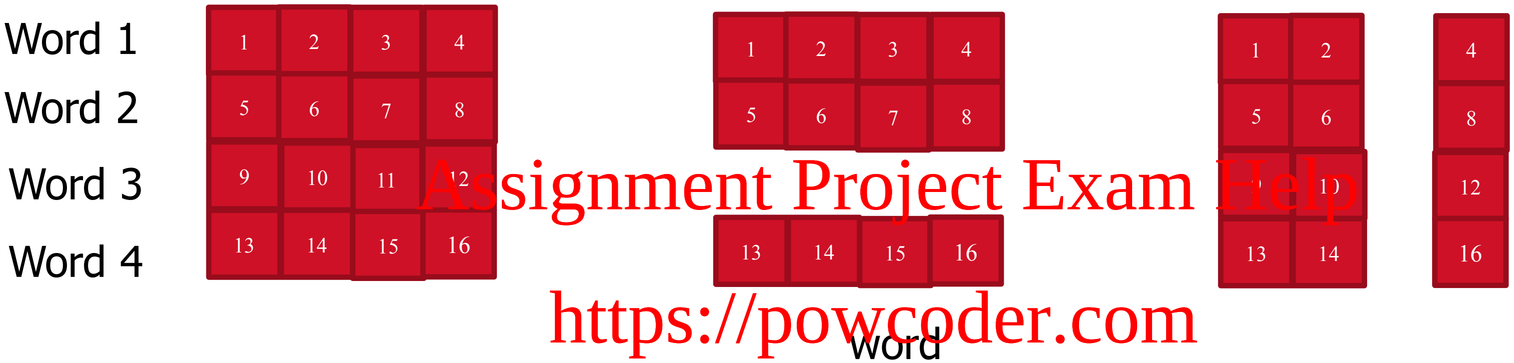
› audio chunks divided into smaller units, e.g. four 5 msec units per 20 msec audio chunk

› packet contains small units from different chunks

› if packet lost, still have *most* of every original chunk

› no redundancy overhead, but worse delay performance

VoiP: recovery from packet loss (cont'd)



e.g., word missing e.g. syllable missing, acceptable

Subjective feeling is improved

Real-time Conversational Applications

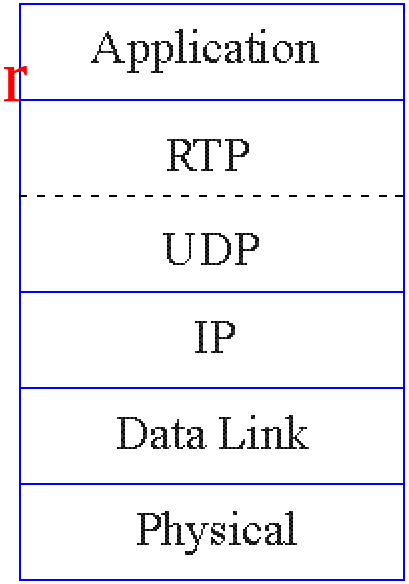
Real-Time Protocol (RTP)

- RTP specifies packet structure for packets carrying audio, video data
- RFC 3550
- RTP packet provides
  - payload type identification
  - packet sequence numbering
  - time stamping
- RTP runs in end systems
- RTP packets encapsulated in UDP segments
- interoperability: if two VoIP applications run RTP, they may be able to work together

RTP runs on top of UDP

RTP libraries provide transport-layer interface that extends UDP:

- port numbers, IP addresses (already existing)
- payload type identification
- packet sequence numbering
- time-stamping



RTP example

example: sending 64 kbps PCM µ-law encoded voice over RTP

PCM: Pulse-code modulation: a method used to digitally represent sampled analog signals

µ-law: Special quatization

Sample rate 8000samples/second

Quantization 8bit/sample

application collects encoded data in chunks, e.g., every 20 msec = 160 bytes in a chunk

- audio chunk + RTP header form RTP packet, which is encapsulated in UDP segment

- RTP header indicates type of audio/video encoding in each packet
  - sender can change encoding during conference
- RTP header also contains sequence numbers, timestamps

RTP and QoS

- RTP does *not* provide any mechanism to ensure timely data delivery or other QoS guarantees

- RTP encapsulation only seen at end systems (*not* by intermediate routers)

  - routers provide best-effort service, making no special effort to ensure that RTP packets arrive at destination in timely manner

RTP header

payload type	*sequence number type*	*time stamp*	*Synchronization Source ID (SSRC)*	*Miscellaneous fields*

- payload type (7 bits): indicates type of encoding currently being used. If sender changes encoding during call, sender informs receiver via payload type field
  - Payload type 0: PCM μ-law, 64 kbps
  - Payload type 3: GSM, 13 kbps
  - Payload type 7: LPC, 2.4 kbps
  - Payload type 26: Motion JPEG
  - Payload type 31: H.261
  - Payload type 33: MPEG2 video
- sequence # (16 bits): increment by one for each RTP packet sent
  - detect packet loss, restore packet sequence

RTP header

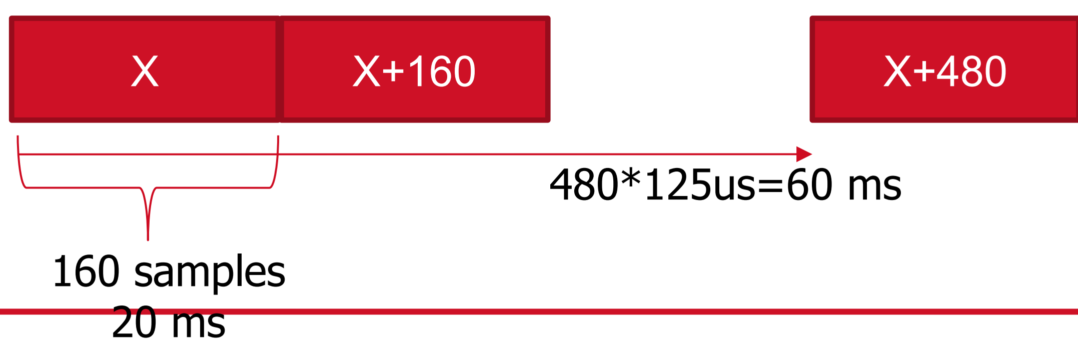
payload type	sequence number type	time stamp	Synchronization Source ID (SSRC)	Miscellaneous fields

› *timestamp field (32 bits long):* sampling instant of first byte in this RTP data packet

- for audio, timestamp clock increments by one for each sampling period (e.g., each 125 usecs for 8 KHz sampling clock)

- if application generates chunks of 160 encoded samples (20ms), 20ms/125us=160

- timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.



RTP header

payload type	sequence number type	time stamp	Synchronization Source ID (SSRC)	Miscellaneous fields

- sequence # + *timestamp:* ***knows new spurts***
- *SSRC field (32 bits long):* identifies source of RTP stream. Each stream in RTP session has distinct SSRC

SIP: Session Initiation Protocol [RFC 3261]

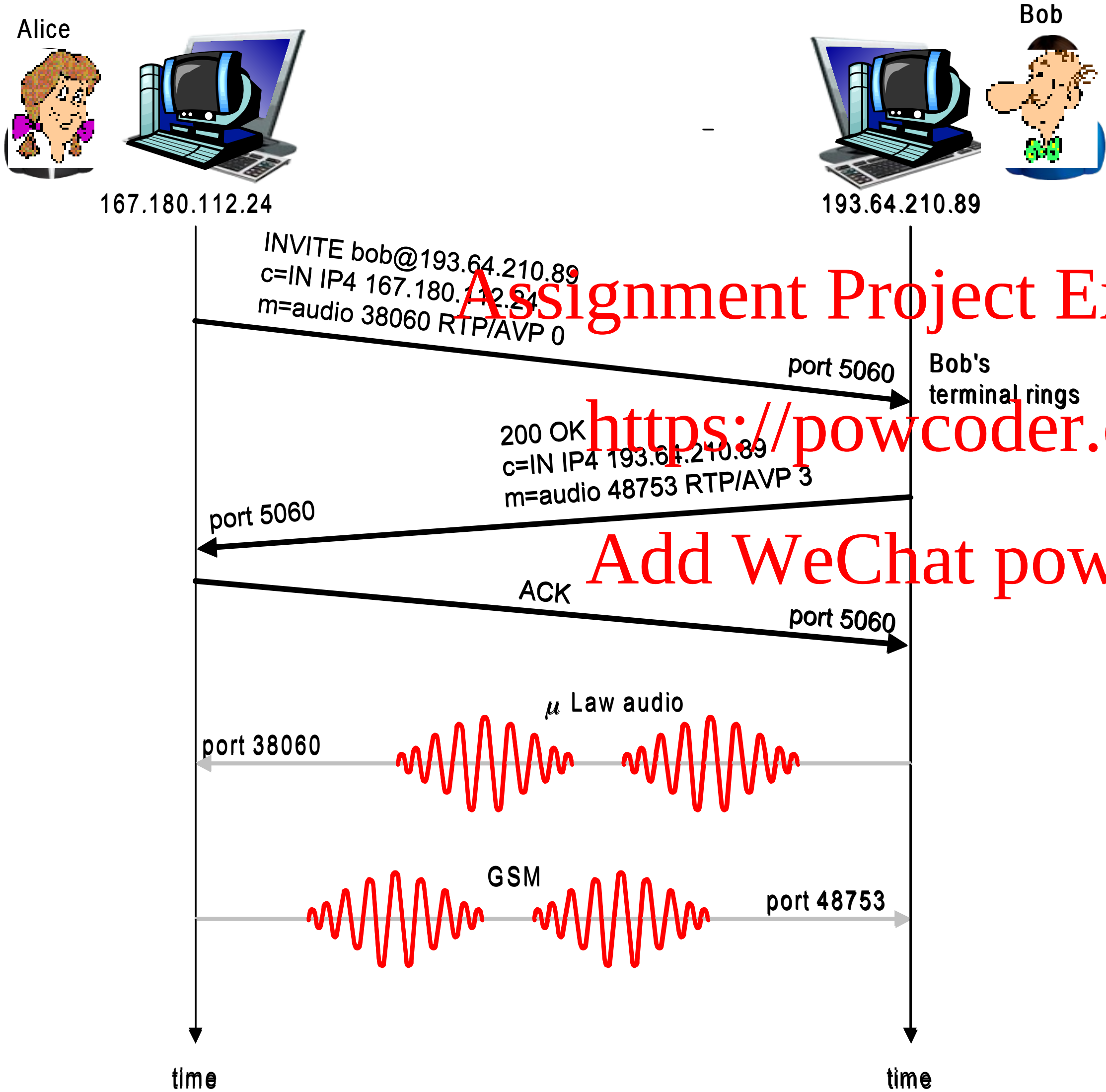
long-term vision:

- all telephone calls, video conference calls take place over Internet
- people identified by names or e-mail addresses, rather than by phone numbers
- can reach callee *(if callee so desires),* no matter where callee roams, no matter what IP device callee is currently using

SIP services

- SIP provides mechanisms for call setup:
  - for caller to let callee know she wants to establish a call
  - so caller, callee can agree on media type, encoding
  - to end call
- determine current IP address of callee:
  - maps mnemonic identifier to current IP address
- call management:
  - add new media streams during call
  - change encoding during call
  - invite others
  - transfer, hold calls

Example: setting up call to known IP address



- Alice's SIP invite message indicates her port number, IP address, encoding she prefers to receive (PCM μlaw)
- Bob's 200 OK message indicates his port number, IP address, preferred encoding (GSM)
- SIP messages can be sent over TCP or UDP; here sent over RTP/UDP
- Default SIP port # is 5060
- Actually, Bob and Alice talks simultaneoulsy
- SIP is out-of-band

Setting up a call (cont'd)

- codec negotiation:
  - suppose Bob doesn't have PCM μlaw encoder
  - Bob will instead reply with 606 Not Acceptable Reply, listing his encoders. Alice can then send new INVITE message, advertising different encoder

- rejecting a call
  - Bob can reject with replies "busy," "gone," "payment required," "forbidden"

- media can be sent over RTP or some other protocol

Name translation, user location

- caller wants to call callee, but only has callee's name or e-mail address.
- need to get IP address of callee's current host:
  - user moves around
  - DHCP protocol (dynamically assign IP address)
  - user has different IP devices (PC, smartphone, car device)
- result can be based on:
  - time of day (work, home)
  - caller (don't want boss to call you at home)
  - status of callee (calls sent to voicemail when callee is already talking to someone)

SIP registrar

- one function of SIP server: registrar
- when Bob starts SIP client, client sends SIP REGISTER message to Bob's registrar server

register message:


```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

SIP proxy

- another function of SIP server: *proxy*
- Alice sends invite message to her proxy server
  - contains address sip:bob@domain.com
  - proxy responsible for routing SIP messages to callee, possibly through multiple proxies
- Bob sends response back through same set of SIP proxies
- proxy returns Bob's SIP response message to Alice
  - contains Bob's IP address
- SIP proxy analogous to local DNS server

THE UNIVERSITY OF SYDNEY

SIP example: alice@umass.edu calls bob@poly.edu